CONTROL YOUR DEVICES FROM A WEB PAGE

Of all of the millions of pages on the web, most are hosted by large servers or, at minimum, desktop PCs, but these aren't the only computers that can function as web servers. Even very small devices can serve web pages on request, including pages that display real-time information and respond to user input. The pages can either be available within a local network or an Internet connection can be added to make the pages available to anyone on the Internet.

The Device Controller described in this article is a small circuit board that controls external circuits and also functions as a web server. Users can access a web page to monitor and control the Device Controller's circuits. The Device Controller is based on a TINI module, which contains a CPU (central processing unit), networking support, and a Java runtime environment which enables the TINI to run Java programs. The TINI was developed by Dallas Semiconductor, a subsidiary of Maxim Integrated Products. The Device Controller's web page displays a virtual control panel (Figure 1). From this page, users can click buttons to turn two LEDs on and off. After clicking a button, the browser's web page updates to match the states of the LEDs.

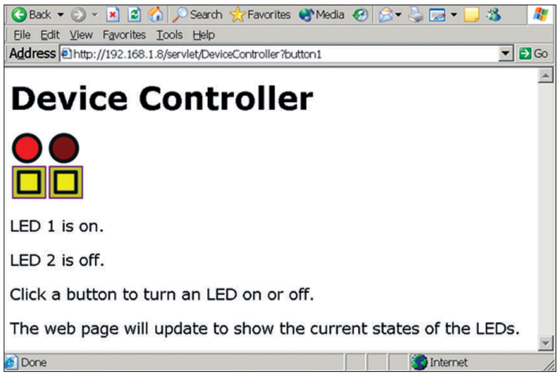
In a similar way, you can program a TINI to monitor and control other circuits or processes, use a web page to enable users' data requests from the TINI, and control the TINI's circuits.

About the TINI

Two options for the TINI module are the DSTINIm400 ($67.00) from Dallas Semiconductor (Figure 2) and the TStik ($99.00) from Systronix (Figure 3). The CPU in both boards is a Dallas Semiconductor DS80C400 Network Microcontroller. Each module plugs into an evaluation board that contains components and connectors for a power supply, RS-232 serial port, and Ethernet. The TStik has an on-board Ethernet transceiver, while the DSTINIm400's Ethernet transceiver is on the evaluation board (DSTINIs400). Creating the Device Controller software, compiling it, and deploying it to the TINI requires a variety of software tools. All are free downloads.

The TINI Software Developers Kit (SDK), available from Dallas Semiconductor, includes the TINIOS operating system and the Java Virtual Machine (JVM) that executes Java programs in the TINI. Dallas Semiconductor also provides the JavaKit utility for configuring the TINI over its RS-232 serial port.

FIGURE 1. The Device Controller's web page is a virtual control panel that enables users to toggle LEDs on the TINI.



Address http://192.168.1.8/servlet/DeviceController?button1

Device Controller

LED 1 is on.

LED 2 is off.

Click a button to turn an LED on or off.

The web page will update to show the current states of the LEDs.

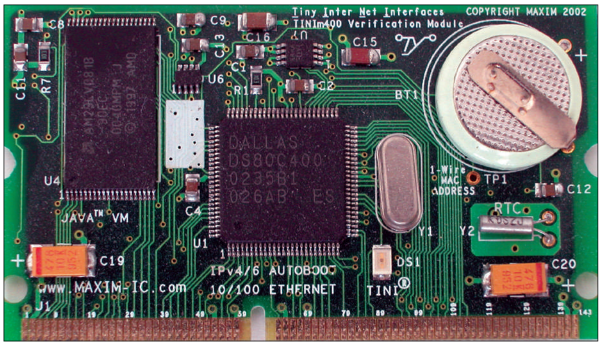
To compile Java programs for a TINI, you can use just about any Java compiler and Java development system, including the compiler in the free Java Development Kit (JDK) from Sun Microsystems. I use Borland's JBuilder environment, which includes a compiler and graphical interface for developing. A free Personal Edition is available. The Ant build tool and the TINI-specific add-on, TiniAnt, are useful for compiling and deploying code to the TINI.

A Java application, called a servlet engine, enables the TINI to run Java programs, called servlets, which can serve web pages that contain real-time information and respond to mouse clicks and other user input. The servlet engine I used for the Device Controller is Shawn Silverman's Tynamo web server, which is free for personal use.

My website at **www.Lvr.com** contains the complete DeviceController servlet code and instructions for running it. For loading files into the TINI and testing the web server on a network, you'll need a PC with an RS-232 serial port and an Ethernet network port.

Why use Java? The Java language is designed from the ground up for use in networking applications and the support built into the TINI's software greatly simplifies network programming. If you're not an experienced Java programmer, an introductory text will get you started. The TINI supports an older, simpler, but still very capable, JDK distribution (1.1.8). If you prefer to program in C instead of Java, my website also has a Device Controller project written in C for Rabbit Semiconductor's RabbitCore modules.

FIGURE 2. Dallas Semiconductor's DSTINIm400 module contains a DS80C400 Network Microcontroller and can run Java programs.



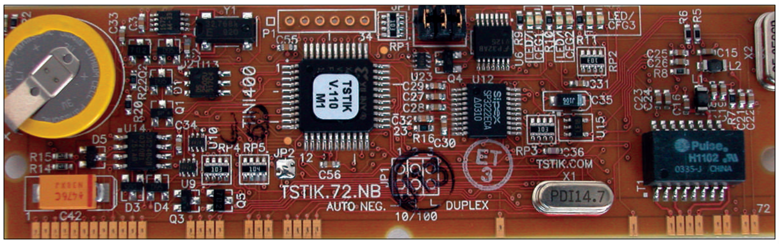
The Web Page

Listing 1 is the HTML source code of the page a browser might receive on requesting the Device Controller's web page. The text between angle brackets (<>) contains HTML code that tells the web browser how to display the text and images on the page.

The page includes four images. Ledon.gif and ledoff.gif are images of lit and unlit LEDs. The images reflect the states of the Device Controller's LEDs when the page was requested. The other two images are identical switches, or buttons (button.gif), which users click to toggle the LEDs.

When a user clicks a button's image, the browser's computer uses the HTTP (hypertext transfer protocol) to send a GET request over the network to the Device Controller. If the user clicks the first LED's button, the request contains the text /servlet/DeviceController?button1. The /servlet/DeviceController portion of the request tells the Device Controller to run the DeviceController servlet. The text following the question mark tells the Device Controller

FIGURE 3. Systronix's TStik also contains a DS80C400 Network Microcontroller and uses the popular 72-contact SIMM connector.



LISTING 1. The HTML code for Figure 1's web page includes links to images that match the states of the LEDs.

```
<html>
<head>
  <title>Device Controller </title>
</head>

<body>
<h1> Device Controller Demo</h1>

<table>
<tr>
  <td><img src ="ledon.gif" ></td>
  <td><img src ="ledoff.gif" ></td>
</tr>

<tr>
  <td>
  <a href="/servlet/DeviceController?button1">
    <img src="button.gif"></a>
  </td>
  <td>
  <a href="/servlet/DeviceController?button2">
    <img src="button.gif" ></a>
  </td>
</tr>

</table>

<p>LED 1 is on.</p>
<p>LED 2 is off.</p>
<p>Click a button to turn an LED on or off.</p>
<p>The web page will update to show the current states of the LEDs.</p>

</body>

</html>
```

LISTING 2. The servlet's doGet() method services HTTP requests for the web page.

```
/**
   * Responds to HTTP GET requests.
   *
   * @param request the HTTPServletRequest object
   * @param response the HttpServletResponse object
   *
   * @throws ServletException
   * @throws IOException
   */
  public void doGet(HttpServletRequest request, HttpServletResponse
response)
      throws ServletException, IOException
  {
  //The query string sent by the client tells which button
  // was clicked on the web page.
    String query = request.getQueryString();

    // Read the current state of the LEDs (0=on, 1=off).
    // and toggle the LED named in the query string.
    boolean led1On;
    if ("button1".equals(query)) {
      System.out.println("Button 1 was clicked");

      // Toggle the LED.
```

Listing 2. continued ...

```
 led1On = toggle(led1);
   } else {

     // Don't toggle the LED.
     // Read the LED's state and set led1On true if the state is 0
     // (on) and false if the state is 1 (off).
     led1On = (led1.readLatch() == 0);
   }

   boolean led2On;
   if ("button2".equals(query)) {
     System.out.println("Button 2 was clicked");

     // Toggle the LED.
     // A logic low turns the LED on.
     led2On = toggle(led2);
   } else {

     // Don't toggle the LED.
     // Read the LED's state and set LED2 On true if the state is 0
     // (on) and false if the state is 1 (off).
     led2On = (led2.readLatch() == 0);
   }

   // Set the images and text to match the LEDs' current states
   // If led1State (led2State) is true, LED1 (LED2) is off.
   // If led2State (led2State) is false, LED1 (LED2) is on.
   String led1Image;
   String led1State;
   if (led1On) {
     led1Image= "/ledon.gif";
     led1State = "on";
   }
   else {
     led1Image = "/ledoff.gif";
     led1State = "off";
   }

   String led2Image;
   String led2State;
   if (led2On) {
     led2Image= "/ledon.gif";
     led2State = "on";
   }
   else {
     led2Image = "/ledoff.gif";
     led2State = "off";
   }

   // Return the web page to the client.
   SendWebPage (response, led1Image, led2Image,
             led1State, led2State);

 } //end doGet
```

which button the user clicked (button1 or button2).

Serving the Web Page

On receiving a request to run the DeviceController servlet, the TINI executes the doGet() method found in Listing 2. One of the parameters passed to the method is an HttpServletRequest object that contains information about the request. If the user has clicked a button, the object's getQueryString() method contains either "button1" or "button2" to indicate which button was clicked.

If the query string contains button1 or button2, a call to the toggle() method in Listing 3 toggles the state of the corresponding LED. Port bits on the DS80C400 control the LEDs. Led1 is controlled by Port 5, bit 4, and led2 is controlled by Port 5, bit 5. Figure 4 shows the circuits.

On the DSTINIs400 board, the bits are accessible from header J21. On the TStik, the bits are contacts 29 and 30 on the SIMM connector. I chose these bits, in part, because they're easily accessible. The bits also function as part of the SPI interface, which isn't available if you use the bits to control the LEDs. You can program the TINI to control any other I/O bits or peripherals you wish.

In the servlet code, led1 and led2 are BitPort objects in the TINI-specific class com.dalsemi.system.BitPort. The class's readLatch() method returns the last value written to a bit. The set() method sets a bit's value to 1 and the clear() method sets a bit's value to 0.

In Listing 2, after toggling the requested LED, the TINI sets the values of four variables that display the states of the LEDs as images and text on the web page. The variables led1Image and led2Image each contain a file name ("ledon.gif" or "ledoff.gif"), depending on the state of the corresponding LED. In a similar way, the led1State and led2State variables each contain the text "on" or "off."

A call to the application's SendWebPage() method sends a web page that displays images and text that match the states of the LEDs on the TINI. A series of out.print statements writes the web page's contents to the ServletOutputStream object out. The browser that requested to run the servlet receives the web page in an HTTP response.

This statement in the SendWebPage() method sends the contents of the page's <head> section, which contains the page's title:

```
out.print("<head><title>DeviceController</title>
</head>"
```

In a similar way, out.print statements send the rest of the HTML code that makes up the web page.

In sending the page, the servlet inserts the file names and text that match the states of the LEDs. This statement places the contents of the led2Image variable (either "ledon.gif" or "ledoff.gif") on the web page:

```
out.print(led2Image);
```

This statement places the contents of the led1State variable (either "on" or "off") on the web page:

```
out.print (led1State);
```

LISTING 3. The servlet's toggle() method toggles the LED at the named port bit.

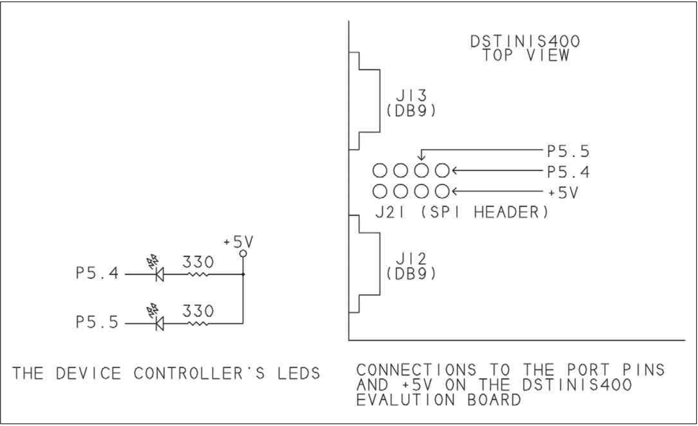
```
/**
 * Toggles the specified bitPort bit and returns the new state.
 *
 * @param bitPort a bit that controls an LED.
 */

private static boolean toggle(BitPort bitPort) {
  if (bitPort.readLatch() == 0) {

    // If it's 0, set the bit to turn the LED off and return false.
    bitPort.set();
    return false;
  } else {

    // If it's 1, clear the bit to turn the LED on and return true.
    bitPort.clear();
    return true;
  }
} // end toggle
```

FIGURE 4. A DSTINIm400 or TStik can control these LEDs.


DSTINIS400
TOP VIEW
J13
(DB9)
P5.5
P5.4
+5V
J21 (SPI HEADER)
J12
(DB9)
+5V
P5.4
330
P5.5
330
THE DEVICE CONTROLLER'S LEDS
CONNECTIONS TO THE PORT PINS AND +5V ON THE DSTINIS400 EVALUTION BOARD

Resources

Ant
(Ant build tool)
www.jakarta.apache.org

Borland
(JBuilder Java environment and compiler)
www.borland.com

Dallas Semiconductor
(DSTINIm400 module)
www.dalsemi.com

Rabbit Semiconductor
(RabbitCore modules)
www.rabbitsemiconductor.com

Sun
(Java Development Kit)
www.java.sun.com

Systronix
(TStik module)
www.systronix.com

TiniAnt
(TINI add-on for Ant)
tiniant.sourceforge.net

Tynamo
(TINI web server and servlet engine)
www.tynamo.com

Running the Web Server

Running the servlet requires a series of steps to build the application webserver.tini, deploy webserver.tini and related files to the TINI module, and run the application. The webserver.tini file contains the code for both the Tynamo web server and the DeviceController servlet.

These are the steps to follow:

1. Download the needed files. The DeviceController files are in devicecontroller.zip, available from **www.Lvr.com** The archive file contains the HTML file DeviceController_readme.htm, which has links to the other files required to build and deploy the Device Controller. Download these as well.

2. Building and deploying webserver.tini uses several configuration files. These are included in devicecontroller.zip. You must edit two of these files with information specific to your system.

The build.properties file contains the locations of the TINI SDK and the DeviceController servlet. Edit these lines to match the locations on your development PC:

```
# The location of the TINI SDK:
tini.path=/tini1.11

# The location of the servlet (DeviceController.java):
src.paths=/myservlets
```

Use forward slashes (/) as separators in the paths, even under Windows.

The deploy.properties file contains information about the TINI. Edit these lines to match the information for your TINI:

• The TINI's IP address:

```
deploy.server=192.168.1.9
```

• The TINI's user ID and password:

```
deploy.userid=root
deploy.password-tini
```

You can view and set the TINI's IP (Internet protocol) address using the TINI's ipconfig command. In JavaKit, type ipconfig to view the current settings and type ipconfig help to view the options you can set.

3. To build webserver.tini, run the Ant build tool by opening a command-prompt window on your development PC, changing to the directory where you stored the Tynamo download, and entering ant on the command line. The Ant tool uses the information in the configuration files to find out what to build and where to find the files.

4. Copy the files ledon.gif, ledoff.gif, and button.gif to the http-root directory in Tynamo's home directory.

5. To deploy the web server to the TINI, connect the TINI to your development PC via Ethernet. For a direct connection, attach a crossover cable directly from the TINI's Ethernet (RJ-45) connector to the PC's Ethernet connector. If your PC connects to an Ethernet repeater hub or switch, attach a straight-across cable between the TINI and an available port on the repeater hub or switch. At a command prompt on the development PC, type ant deploy. This causes the web server's files to transfer to the TINI via the File Transfer Protocol (FTP).

Making the Device Controller Available on the Internet

To access the Device Controller's web page from the Internet, you need to have an Internet account that permits hosting a server and network-security settings that enable the TINI to receive connection requests on port 80 (the default port for HTTP) or another port you specify.

Hosting a server may require a business account for Internet access. Accounts offered to home users typically forbid hosting servers because a server may draw more traffic than the provider can support at home-user prices. Some providers block incoming requests to open connections to port 80.

If the TINI accesses the Internet via a dial-up connection, you will, of course, need a phone line that's available to the TINI whenever you want the Device Controller to be on line.

It is wise to use a firewall to protect your computers that access the Internet. A firewall may be software running on a PC in the local network or a piece of dedicated hardware. If your TINI is behind a firewall, you'll need to configure the firewall to allow the TINI to receive incoming HTTP requests. One way to achieve this is to set up the firewall so that all requests to open a connection to Port 80 are routed to the TINI.

To access the TINI from the Internet, you need to know its public IP address. If you use a firewall with Network Address Translation (NAT), the firewall will use a single public IP address for all Internet communications, converting between the public and local IP addresses as needed. The firewall should have a configuration page that displays its public IP address.

From a browser, you access the Device Controller the same way you would in a local network, except that you must use a public IP address. If the address is 192.0.2.3, you would enter this in the browser's address box:

http://192.0.2.3/servlet/DeviceController

If the TINI's IP address has an assigned domain name, you can also access the Device Controller using the domain name:

www.example.com/servlet/DeviceController

In any case, it's best to check out the Device Controller on a local network before attempting to access it from the Internet.

6. To run the web server in Windows' Hyperterminal or a similar application, create a Telnet connection with these settings:

Host Address = the TINI's IP address
Port = 23
Connect = TCP/IP
Click Call to connect to the TINI and run the web server by entering this text in the Telnet window:

```
source web/bin/WebServer
```

When the server is running, the Telnet Window will display this text:

```
HttpServer: Server started.
```

The TINI is now ready to run the DeviceController servlet.

Accessing the Web Server

On a computer in the same local network as the TINI, in the address text box of a web browser, enter http://, followed by the TINI's IP address, /servlet/, and the servlet's name. For example, if the TINI's IP address is 192.168.1.9, enter this:

```
http://192.168.1.9/servlet/Device
Controller
```

You should then see the Device Controller's web page with images and text that match the states of the LEDs on the TINI module. Click a button and the corresponding LED on the TINI will toggle and the browser will update with a new web page that reflects the change.

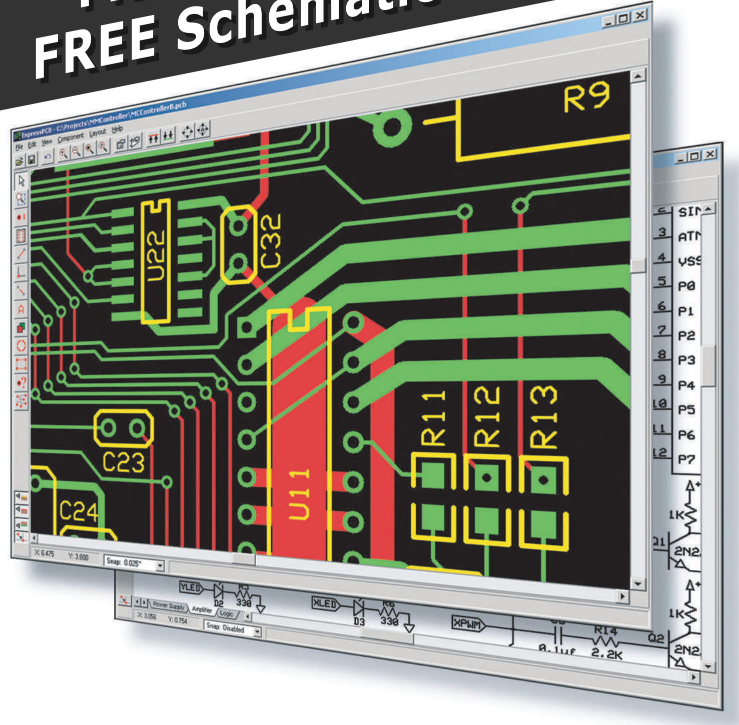
This basic application can serve as a template for TINIs that perform other monitoring and control tasks and serve web pages which enable users to view and control the TINI's activities.

My website (**www.Lvr.com**) has links to more documentation about the TINI and Tynamo web server and how to use them, as well as the complete source code for the Device Controller. **NV**

Jan Axelson is the author of *Embedded Ethernet and Internet Complete*, *USB Complete*, and other books about computer interfacing. For more about computer interfacing via Ethernet and other ports, visit Jan's website at **www.Lvr.com**

$51 For 3 PCBs

FREE Layout Software!
FREE Schematic Software!



01 Download our FREE CAD software
02 Design your two or four layer PC board
03 Send us your design with just a click
04 Receive top quality boards in just days

expresspcb.com